

Blue Sky: A side-scroller computer game

GRADUATE PROJECT REPORT

Submitted to the Faculty of
the School of Engineering and Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

By

Kien H. Dinh

Fall 2014

Committee Members

Dr. Scott A. King
Committee Chairperson

Dr. David Thomas
Committee Member

ABSTRACT

In recent years, computer graphics and game programming has become an immensely popular topic. There exists a massive and complex industry behind the creation of video games that brings to the world something innovative, creative and exciting where dreams and fairy tales come true. The goal of any good video game is to bring to life a story, and to immerse the player as a main character. This project describes the design and development process of such a digital game, titled Blue Sky in which the player guides a powerful atomic helicopter in the sky through several stages of opposing enemy forces.

Table of Contents

ABSTRACT.....	1
Table of Contents	2
LIST OF FIGURES	4
BACKGROUND AND RATIONALE.....	5
1.1: Related Work.....	6
1.2: Microsoft DirectX SDK.....	7
1.3: Game story.....	8
2: SYSTEM DESIGN	9
2.1: Game design.....	9
2.2: Animation sprite.....	10
2.2.1: Image	10
2.2.2: Reading image.....	11
2.2.3: Animation.....	11
2.3: Game loop.....	11
2.4: Assets.....	12
2.4.1: System entity-relationship diagram.....	12
2.4.2: User interfaces	13
2.4.3: Sound and music.....	15
2.4.4: Collision detection.....	17
2.4.5: Class diagram.....	19
2.4.6: Screen graphic.....	26
3: TESTING AND RESULTS.....	28
3.1: Testing.....	28

3.2: Result.....	28
4: FURTURE WORK	32
4.1: Game Modes and Game Stages.....	32
4.2: Game level.....	32
<u>4.3: Developing for mobile devices.....</u>	32
REFERENCES	33
Appendix A: GAME DESIGN	35
Appendix B: USER MANUAL	40
Appendix C: TESTING FORM.....	42

LIST OF FIGURES

Figure 1: Game loop	12
Figure 2: System relationship diagram	13
Figure 3: Collision detection.....	18
Figure 4: User case diagram	19
Figure 5: Response to the question “Do you like game”.....	29
Figure 6: Response to the question “Did you finish the game”	29
Figure 7: Response to the question “How many times you play game”.....	29
Figure 8: Response to the question “Was the game too short, too long or just right?	
.....	29
Figure 9: Response to the question “how much you like materials and/or game pieces”	
.....	30
Figure 10: Response to the question” Do you like the game idea?”	30
Figure 11: Main character.....	34
Figure 12: Ground enemy	31
Figure 13: Fly enemies.....	36
Figure 14: Boss	36
Figure 15: Bullet	37
Figure 16: Explosion.....	37

BACKGROUND AND RATIONALE

The rapid development of computer science and technologies affects every aspect of people's lives, such as learning, living, working, and education. In addition, computer video games are not only just a form of computer software but also a new form of artistic expression thanks to the development of the computer technology. Video games are an important application of computer technology and play a major role in entertainment. It has become accepted by people of different ages. As computer technology is becoming more convenient and accessible, more and more people not only like to play computer games designed by others, but also want to try designing a computer game themselves. Therefore, game development has become an important research area.

A game engine is a framework used to design and development computer games. A game is basically a combination of codes and setting control information. Research by WenFang and Wei (2008) suggests game engines and games share a relationship similar to the car and car engines. With the same kind of car engine, we can make different car models by using different body styles and colors. Similarly, a single game engine can be used to create several different games. Building a car requires many more elements than just an engine though. For example, to build a car, it needs to have car door, car engine, color, glass and so on. A video game also requires many more elements than just the game engine, such as characters, story arcs, and sound effects.

Microsoft Windows uses an internal messaging model to control windows applications. Messages are sent as events occur. Wherever players create an event, a message is sent. For example, when player press a key on the keyboard, the operating system generated a message and put this message in to a message queue. The system

passes all input for an application to the various windows in the application. Each window has a function called a window procedure. The window procedure processes the input and return control to the system.

1.1: Related Work

A side-scrolling video game is one of the most popular technologies to make a video game. Another name of it is side-scroller. There are many kinds of videos games developed by using side-scrolling. The actions in game are viewed from a side-view camera angle. The onscreen characters generally move from the left to right side of the screen to meet an object. The object can be enemies, bombs, bosses...According to Steven Kent(2003), the author of *The Ultimate History of Video Games*, he suggested that the movement from single-screen or flip-screen to scrolling graphics such like evolution of people from the Stone Age to the Bronze Age. It would prove to be a pivotal leap in the video game design.

There are many successful video games developed using the side-scrolling model. Super Mario and Super Mario Bros videos game are two of the most famous side-scrolling. Side-scrolling video games are used in certain role-playing video games. The screen scrolled to a certain point then stop and requires the enemies on screen to be defeated before the player can keep moving on. Another popular way to use the side-scrolling format is that player usually starts with a character that flies from left to right screen and face an ever increasing horde of enemies. Defender is an example for this type of game.

Research by Andrew (2005) support the screen will scroll forward following the speed and direction of the player character and can also scroll backwards to previously

visited parts of a stage. The other way to do this is to have the screen follow the player character but only scroll forwards, not backwards, that mean once something has passed off the back of the screen, and it can no longer be available or visited.

Some side scroller video games have different stages where the screen scrolls forward by itself at a steady rate, and player must keep up with the screen, attempting to avoid obstacles and collect things before they pass off screen. The “Wind Runner” is an example. This is a video game for iPad and iPhone created by WeMade Entertainment CO., LTD. The players have to collect all achievement before it is keep running out of screen.

1.2: Microsoft DirectX SDK

Microsoft DirectX is a collection of application programming interfaces (APIs) handling tasks related to multimedia, especially game programming and video, on Microsoft platforms. The “X” at the end stands for the particular API names. This is a funny story because at first Microsoft thought that” X” should be standing for Xbox to indicate that the console was based on DirectX technology. DirectX is a recommended tool to use in the development of videos game for Microsoft Windows and Microsoft Xbox.

According to the Microsoft Developer Network Library on 2000, the Microsoft DirectX Software Development Kit (SDK) consists of runtime libraries in redistributable binary along with accompanying documentation and headers for use in coding. Runtime libraries are sets of low-level routines which are used by a compiler (Visual Studio is an example for compiler) to appeal to some of the behaviors of a runtime environment, by inserting calls to the runtime library into compiled executable binary. The runtime environment implements the execution model, built-in functions, and other fundamental

behaviors of a programming language. When a program executes a code, it calls the runtime library causing communication between the executable binary and the runtime environment. A runtime library often includes built-in functions for memory management, or for exceptions handling. Therefore, a runtime library is always specific to the platform and compiler.

In general, a video game development will be two teams working together such as a design game team and a programing team. However, I would like to do both of work to help me understand the whole concept of game development. In addition, the SDK is available as a free download. While the runtimes are proprietary, closed-source software, source code is provided for most of the SDK samples. Starting with the release of Windows 8 Developer Preview, DirectX SDK has been integrated into Windows SDK.

1.3: Game story

Far away from Earth, at a planet has name Phillon, the story has begun. This is a war between the government and rebels. The government's military named as the largest city of the Decan: Bygeniou City United (B.C.U). The National Force Union controlled the government and their goal is to unify the planet of Phillon. The rebel's military has placed in the poorest city of the Decan: Arlington, and has name Arlington National Influence (A.N.I).

Michael is a top ranking pilot that has had remarkable results in battle. Not very personable and keeps his distance from other. He is used to be a member of the B.C.U Government, but now, he is standing by the A.N.I faction. His mission is control a helicopter and goes back to Sunken city to restart an AI machine. This mission is very dangerous because Sunken is place where B.C.U enemies are occupied.

2: SYSTEM DESIGN

The narrative of this project is the design and programming of a video game. Many elements are necessary to develop a video game such as music, artwork, graphics, collision detection, and much more. There are many modern video games that look, sound, and behave so well that it can be hard to imagine how you'd begin to design and program something similar. Most of video game details started or come from some basic game programming concepts. The project describes how to start, design, process and put every element together to develop a video game.

BlueSky is a video game supported by the Windows operating system. It is created using Java programming language. The main character will control a helicopter and use bullets and bombs to destroy hostile enemies. The user will use a keyboard to control the main character's actions such as move up, down, left, right and fire bullets. The game also has a support system to help players understand how Blue Sky works.

2.1: Game design

Game design relates to the process of conceptualizing the mechanisms of the game. It is an art combination rules and game mechanics. The design process is an integral step in game development. BlueSky is designed as an action video game. It has a main character piloting a helicopter and five types of enemies. The game also has a special boss enemy. Each player has five attempts to beat the game. The player loses one life when hit by enemy attacks. Please see the appendix A to have more details for BlueSky video game design.

David Brackeen[#] described three kinds of videos game. First, applet games are applications that run on a web browser. One benefit of applet games is that players don't

have to install anything on their computers to play. However, users need an internet connection and a web browser to play these video games. The primary concerns of applet games is network security and a steady network connection. The second game type mentioned by Brackeen [#] is *windowed* games. Contrary to applet games, windowed games do not have security restrictions, and don't run in an internet browser. Windowed games look like a normal Windows applications. BlueSky is a windowed game. The third game type is *full screen* games, which give players total control over the visual presentation in the game.

2.2: Animation sprite

A game's active entities are often encoded as sprites. A sprite is a moving graphical object. A sprite can present player main character or an enemy. A game has several sprites. The sprite properties are move speed and position. On the screen, it can be one or many sprites at the same time. The sprite comes from the image.

2.2.1: Image

Images are the most important part of every game. Raster and vector are two types of basic image formats. A raster image format describes image in terms of pixels with a specified bit depth. A vector image format describes images geometrically and it can be resized. These three formats are GIF, PNG and JPEG. GIF is an image that can have 8-bit color. Moreover, it has high compression for graphics image and PNG image can supersede GIF's function. Therefore, the game does not use GIF image. JPEG is a 24-bit image with high compression for photographic image. The last one is PNG image which has 24-bit color. The image type used in this project is PNG and some JPEG.

2.2.2: Reading image

The *loadImage* class loads an image from a directory. This class can load different formats of images. The images are stored in a local file in a subdirectory *images/*... The image below does describe an example how to load an image from directory. It parses the image and returns an image object (sprite). In Bluesky, the function *loadImage()* loads image from directory and put it into a new sprite. The structure command to load image is *Image name_image = loadImage("file location")* and set to new sprite by *sprite()* class. The background is a PNG format file because needs the quality of pictures.

2.2.3: Animation

When access with several images, it can do the animation. The animation loop is follow four step: update animation, draw to the screen, take a nap, and start over first step. BlueSky is using cartoon-style animation. Animation is where several images are drawn in a sequence to create the illusion of movement. Each image is a frame and each frame is displayed for a certain amount of time. However, frames don't all have to display for the same amount of time. The animation class has three important methods: *addFrame()*, *update()* and *getImage*. The *addFrame()* method adds an image to the animation with a specified time in milliseconds to display. The *getImage()* method gets the image that can be displayed based on how much time has passed. The *update()* method tells the animation that a specified amount of time has passed. *getImage()* gets the animation's current image and if the animation has no image, the return value is null.

2.3: Game loop

Basically, the game loop is core of the game. This runs continuously during gameplay. The game loop follows steps which describe below:

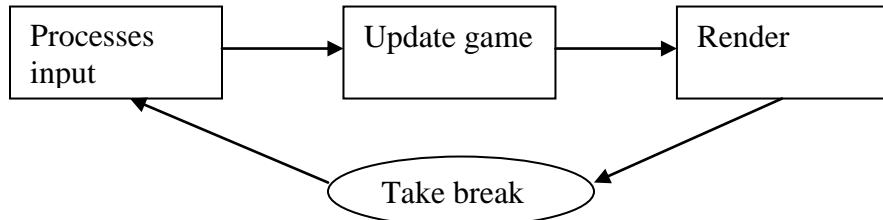


Figure1: Game loop

The bluesky applied these four steps. The variable `isRunning` defined to tell the game loop that it's time to quit. The class `run()` calls `gameInit` and `gameLoop`. The `gameInit` sets full screen mode, initiates and object. The `gameLoop` controls update and draw sprite to the screen. The class `gameLoop` and `gameInit` called again in the main program. What the game update during gameplay is image and sound. The images can be weapon, enemy. Therefore, depend on the input from keyboard, it reflects to the player. Each time player start a new game, this class refresh score and player life times. The `gameLoop` function base on how many time played in the game. The game loop is runs until `stop()` is call. The `stop()` function stops when `isRunning` is false.

2.4: Assets

The assets are the useful to develop the game. The game has several assets such as class diagram, music and picture files. This part describes how to build the game from assets.

2.4.1: System entity-relationship diagram

The system diagram describes the relationship between each functions type in the game. The game has three main function are start game, support player and exit. When

the game starts, player must play and earn as much points as possible. In case players do not know how to play, player can go to help screen to see the introduction.

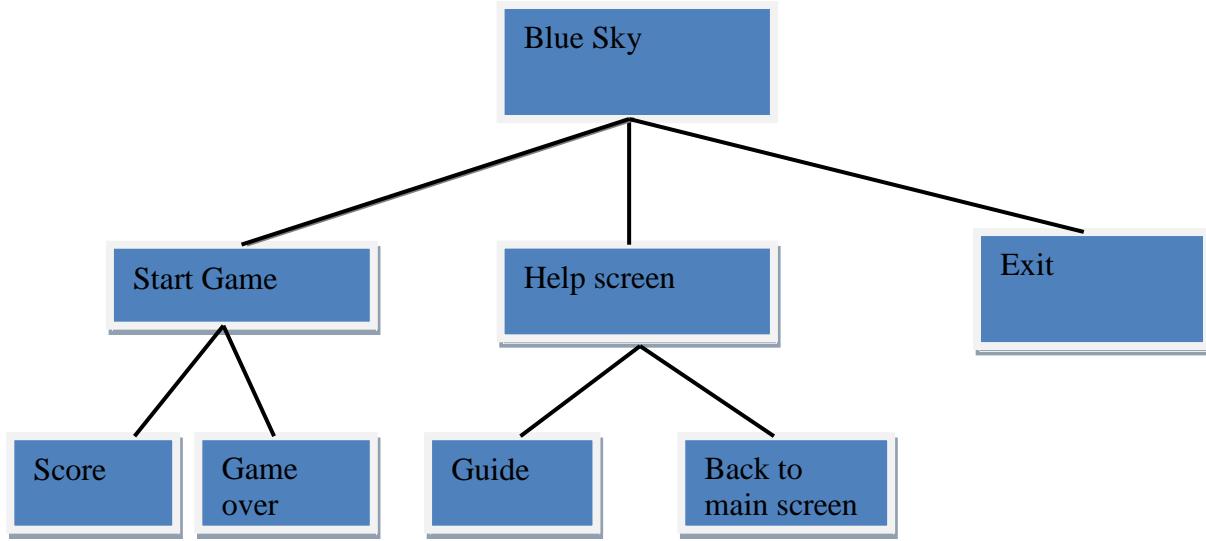


Figure 2: System relationship diagram

The system has three main functions. There are start game, help and exit. First, start game function is at the beginning, when the player starts the game, his mission is gain as much score as possible and kill the enemies also. The player will have five times to play Blue Sky. In addition, the player can earn extra times (Life points) for every 1000 points scored. Second, the game supports the player to understand how the video game works. By using help screen, the player can see the guide of the game and how to control and using keyboard. The third function is exit game.

2.4.2: User interfaces

User interface is the way the external devices process with the game. Bluesky used keyboard as the external device. In this section, it discuss about game events, keyboard and mouse input, how to create and use input manager.

2.4.2.1: Game event model.

There is a support from JAVA called Abstract Window ToolKit (AWT). This is a set of libraries to create graphic user interface. When an event occurs on a particular component, the AWT checks to see if there are any listeners for that event. A listener is an object that receives events from another object. We have different types of listeners for different events. For example, there is *MouseListener*, a mouse input event, or *KeyListener* for keyboard input event. All listeners are interfaces, so any object can be a listener by implementing a listener interface.

2.4.2.2: Keyboard

Keyboard is used to control the main character in Blue Sky. As in many video games, I want to use the arrow key for movement and Control key for firing weapons. What I need is to capture key events. So, I need to create a *KeyListener* and register the listener to receive events.

At first, to create a *KeyListener*, I need an object that implements the *KeyListener* interface. The KeyListener has three methods. There are *keyPressed()*, *keyReleased()* and *keyTyped*. These methods take a *KeyEvent* as a parameter. The KeyEvent object enables users to inspect what key was actually pressed or released. Second, it needs to register the listener to receive events. To do it, I just call the *addKeyListener()* method on the component that I want to receive key events on.

2.4.2.3: Create and use input manager

Until now, the game has mouse and keyboard input. It need to create an input manager called *InputManager*. That mean put mouse and keyboard input together. What does the game want *InputManager* to do? At first, it can handle all key and mouse events. Secondly, it can save the events so that users can query them precisely when they

want to. Thirdly, need to map keys to have generic game actions. Finally, I want it so users can configure keyboard.

To do it, I used *GameAction* class and *InputManage* class. The *GameAction* class has method *isPressed()* for keys and *getAmount()* to see how the mouse has moved. The *InputManage* class makes mapping keys and mouse events to come to *GameAction* class. At the time when key is pressed, it will check to see whether a *GameAction* is mapped to that key and calls *GameActionSpress()* method. The array *GameActions* is created so that each index in the array corresponds to a virtual key code. The *GameActions* array has a length of 600 because most key codes have a value smaller than 600.

2.4.2.4: The game pausing.

We will keep drawing to the screen even if the game is paused. However, we need to draw a “pause” screen or maybe a “pause” message. We can choose between “ESC” and “P” to pause the game. When we pause the game, there are two things that happen. Firstly, the object and animations aren’t updated. Secondly, input is ignored excluding the key to un-pause the game.

2.4.3: Sound and music

The combination of sound, music and video game should synchronize and parallel. Sound effects are an important part of a game. In fact, when people are playing the game; they hear the sounds but do not really notice them. Sound is created when something vibrates through a medium. The medium is air and vibrates is your computer’s speaker. This section discuss about how to create a sound, how to play a sound and, how to set the sound into the game.

In Blue Sky, there are two different types of sounds: background music and audio effect. The first sound is background music. The problem with background music is that

it needs memory and loading time. High-quality stereo sampled audio may require about 10MB per minute of sound, but a typical MIDI sequence may need less than 10 KB. A long piece of music should be encoded as a MIDI sequence to save disk space and loading time. However, we just need one background sound at a time. The second is audio effects. Audio effects can be explosions, when player dies, earns point, hits bullets... Not like the background music, the audio effects may need several sound effects at the same time or overlap in time.

Even though we may not find background music in every game that we play, but we cannot deny that it is an important part of game. There are different music moods for different type of games. For example, the action game should play fast-paces music while some studies game can play with slow music. In addition, music also explains character status and changes the player's emotions. For example, dramatic music can be played when game system increase difficult levels or when player fight with boss. It needs to decide on the type of music that we want to use.

2.4.3.1: Creating and sound format.

There are several tools to create record and edit sounds.

The sound system supports sound file formats: AIFF, AU and WAV. It can control sound formats with 8 or 16 bit samples with sample rates from 8000Hz to 48,000Hz. In addition, it can play mono and stereo sound.

2.4.3.2: Playing sound.

To play a sound, we need to load sound from directory and then put it in a line.

Firstly, we can load a sound file by using *AudioSystem* class. The *AudioSystem* class provides *getAudioInputStream()* methods to open an audio file from the system. It is interesting that this method can open a sound file from Internet. This method will return

an object. It will put the object to *AudioInputStream*.

The *AudioFormat* class provides a way to get information about the format of the sound such as the sample rate and number of channels. For example, a three-second-long sound with an audio format of 16-bit samples, stereo, 44,100Hz would be $44,100 \times 3 \times 4$ bytes, or about 517KB.

The game uses a line to put all of the sounds together. A line is an interface to send or receive audio from the sound system. The lines is use to send sound samples to the sound system to play or receive sound.

Secondly, the lines are created by using *AudioSystem.getLine()* method. It passes this method a *line.Info* object which specifies the type of line want to create. The *line.Info* has a *DataLine.Info* subclass, this subclass does use to create lines because it contains information on the line's audio format.

The game has sound files and put it in a line. The next step it need to do is playing a sound. *SimpleSoundPlayer* is class to play sound. Firstly, this class loads samples from *AudioInputStream* that we mention before into a byte array. Secondly, this class is copying data from *InputStream* to line. The *getSamples()* method reads from *AudioInputStream* and stores the data in the samples byte array. The *play()* method reads data from *InputStream* to a buffer and then writes the buffer to a *SourceDataLine* and the *SourceDataLine* will play the sound.

2.4.4: Collision detection

The main character cannot go through enemies without anything happening. The problem is how we can describe the detection between two sprites. The sprite could move across several tiles at once and may be located in many different tiles at one time. So, the game need to check every tile .The sprite is currently in an every sprite the tile is going to

be in. That means we need to detect when the player collides with other object, other sprite. In Blue-Sky, the *TestCollision* class will check collision between two objects.

After checking for collision, need to update and check collision for next object. It knows that the object is present by sprite and it can be a bullet or an enemy.

```
1213 // Test collision between 2 obj
1214 public boolean TestCollision(int X1,int Y1,int Width1,int Height1,int X2,int Y2,int Width2,int Height2)
1215 {
1216     Rectangle bound_rect_1 = new Rectangle(X1,Y1,Width1,Height1);
1217     Rectangle bound_rect_2 = new Rectangle(X2,Y2,Width2,Height2);
1218
1219     // Check for collision
1220     if (bound_rect_1.intersects(bound_rect_2))
1221     {
1222         return true;
1223     }
1224
1225     return false;
1226 }
```

Figure 3: Collision detection

Assume that everything that moves in the game is an object; there are three steps to detected collision

Update the object's location

Check for any collisions with other objects or with the environment

If a collision is found, revert the object to its previous location.

Also need to check for a collision after each object moves. In addition, the game needs to move all the objects at first and then check collisions afterward. The objects previous stored in the location with the object.

A sprite must monitor the game environment and the collision is one part of it. The collision processing can be split into two basic categories: Collision detection and collision response. In 2D game, we used the bounding rectangles of the sprites for collision detections. We talked about isolating objects on a grid to limit the number of actual collision test to make. The problem is we don't want to check collisions everywhere on the screen; we just need to check collision to the objects which are near

the main objects, that means we need to reduce the number of object-to-object collision test by doing the test only for object against other object that is in the same cell and surrounding cells. In the Blue Sky, the objects are flying objects, so we do not need to add gravity to each object.

2.4.5: Class diagram.

There are many classes in this program. I will use Unified Modeling Language (UML) to explain how each class works and the relationship between classes. UML is a simple language as a tool for expressing ideas

2.4.5.1: Use case

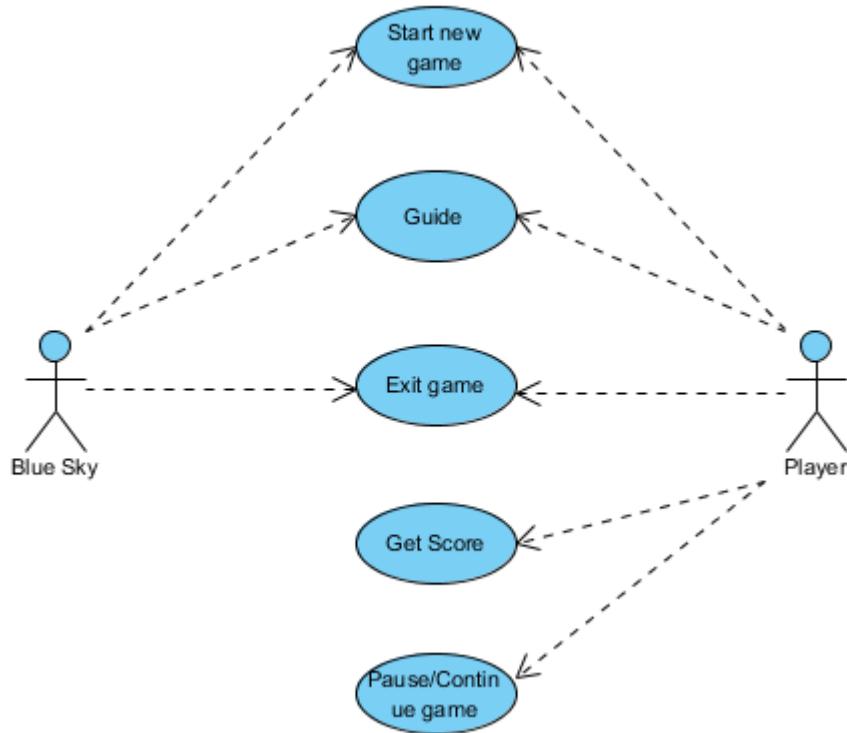


Figure 4: User case diagram

The first notation is bluesky. This bluesky is an object and instance of a class bluesky class. The full UML notation is *bluesky:bluesky* class. The BlueSky provide start a new game, guide and exit game for player. The player uses the options of start and exit

game, views how to play by uses guide, get score or pause/continue the game.

2.4.5.2: Class diagram

For every class in the BlueSky, there will be three areas filled in with enough info to concisely describe the class: class name, member of class and methods.

The first class is *AirCraft()*. The structure of this class is *AirCraft(Sprite _graph, int _speed, long _time_fire,int _X,int _Y,int _width,int _height)*. The proposal of this class is create and handle sprites. The sprite can be enemies or main character. The attributes of an enemy are image which describe by a sprite, move speed, time to fire and location.

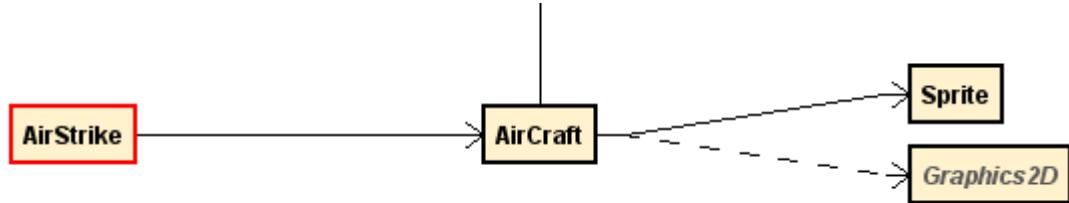
AirCraft	
+void FlyForPlayer(int)	
+void FlyForComputer(int)	
+void FlyForBoss(int)	
+void SetUse(boolean, int, parameter)	
+void SetFirePre(long)	
+void GetFirePre(long)	
+void SetBombPre(long)	
+void GetBombPre(long)	
+void SetTimeFire(long)	
+void SetSpeed(int)	
+int GetWidth()	
+int GetHeight()	
+int getX()	
+int getY()	
+void SetScore(long)	
+void GetScore(long)	

There are details of main functions in this class:

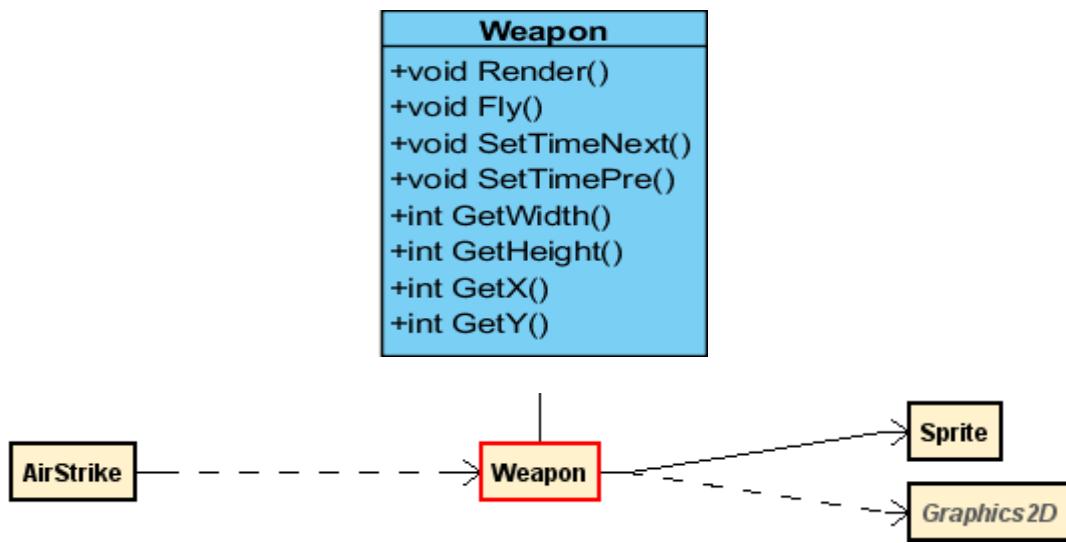
- *Void FlyForPlayer (int direction):* Control movement for player based on direction.
- *Void FlyForComputer():* Control movement for enemeis based on direction
- *Void FlyForBoss :* Control movement for boss.
- *Void Render(Graphics2D g):* This is a subclass of Graphic which defined in java.awt.Component. This class draws the sprite for BlueSky. In addition, it is guaranteed that the object passed to this method can safely be cast to Graphics2D.

- *Void SetSpeed* : setting movement speed for sprite
- *Void SetTimeFire*: Control how many time to fire.

The image below is describle the relationship between the AirCraft and other class

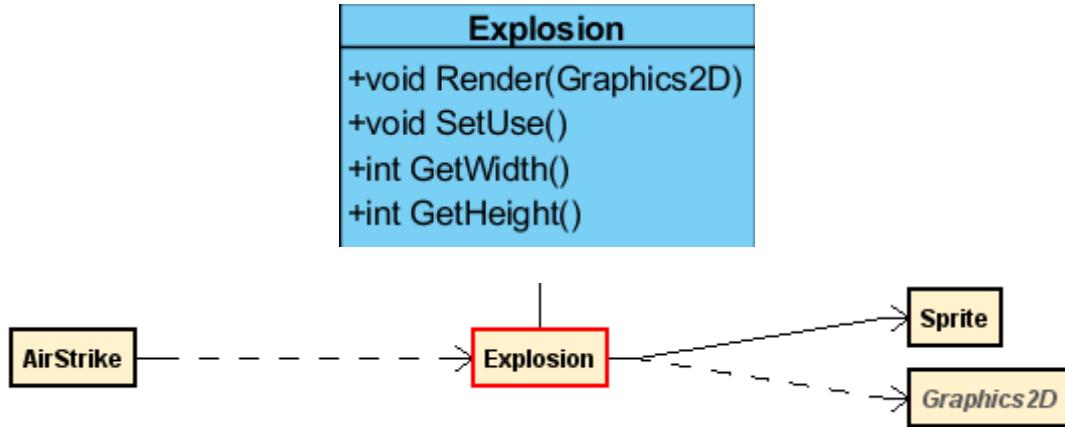


The second class is weapon class. This class will draw and control weapon for main character and enemies also. The constructors of this class is *Weapon(Sprite _grap, int _speed, int _damage,int _X,int _Y,int _width,int _height)*. The weapon atributes are image of weapon such as bullet or bomb, how much speed for it, how much damage it take and location. The class void Render draws weapon to screen like we draw the character and enemies in the first class.

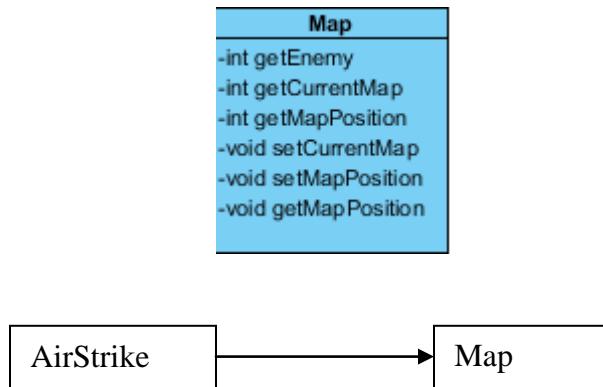


The third class is explosion. What we have now are a main character sprite and a lot of enemy sprites. In addition, they are flying in the game. This class provides a method to create and control the collision between object and object. The constructors is

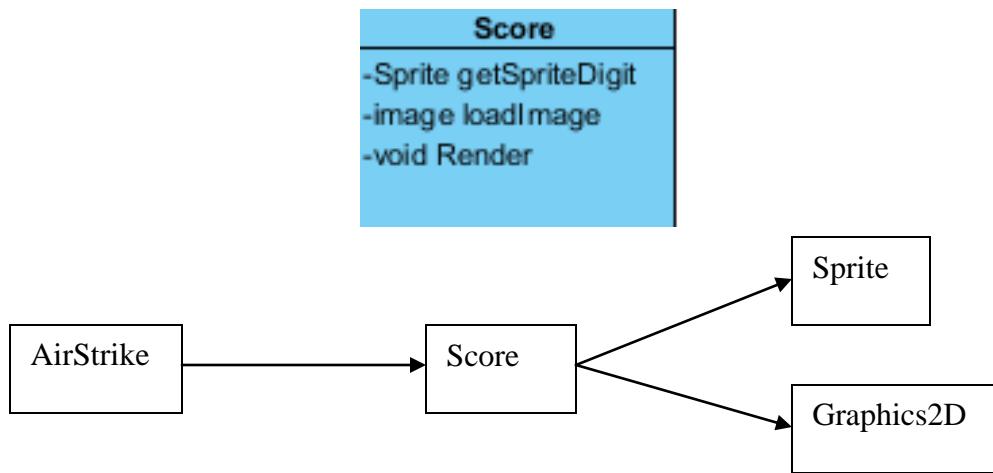
Explosion(Sprite _grap, int _X, int _Y,int _width, int _height).



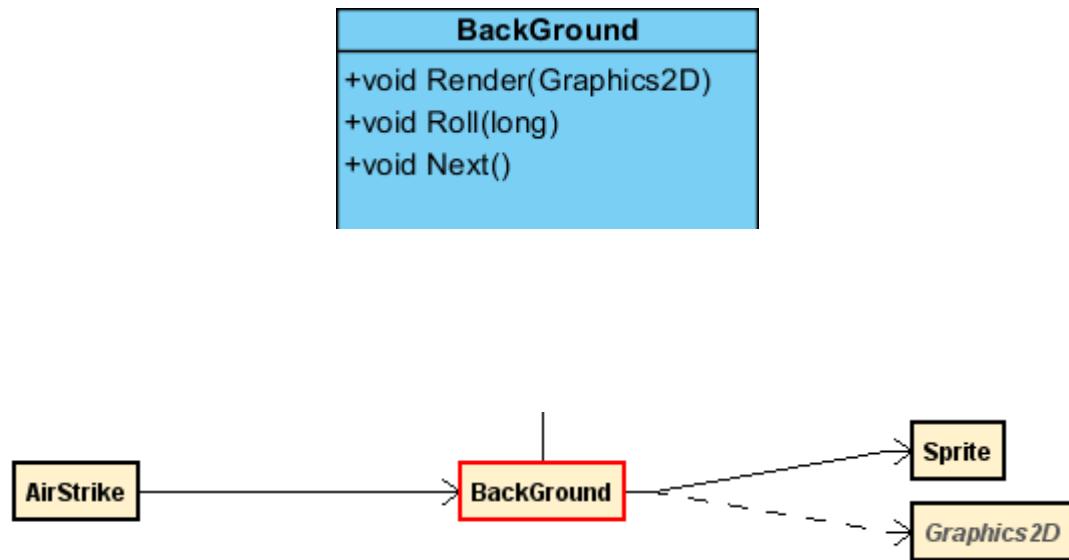
The fourth class is map. This is not public class. This class will get the enemies to the map and setting the map for the BlueSky. The properties of this class are *currentMap* and *mapPosition*. In this class, I define the length of the map and get the enemies. I use a variable *map_length* to define map length and set it as an integer. The enemies will appear to the map base on the current map.



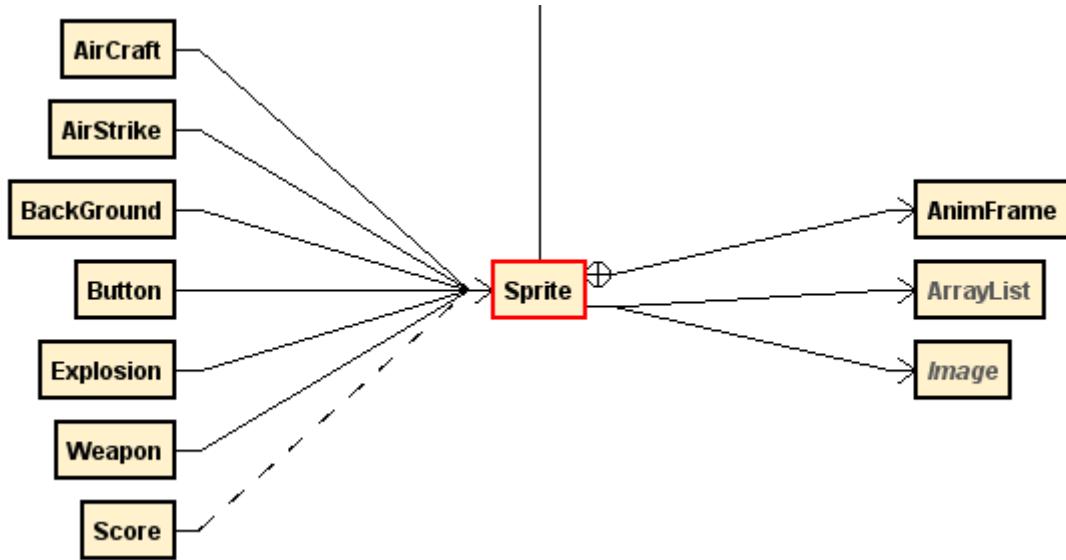
The class score control and calculate the player score. It has three methods: *render*, *loadImage* and *getSpriteDigit*. As I mentioned before, the render class draws the image which load from the *loadImage* methods. The score class loads images from directory folder and adds them to a sprite array. The image is a list of 10 digit numbers.



The class *BackGround* controls and rolls the map. Two main methods in this class is *Render()* and *Roll()*. This class controls how much time to scroll screen base on the elapsed time. In addition, this class control which map layer appears when amount of time has passed.

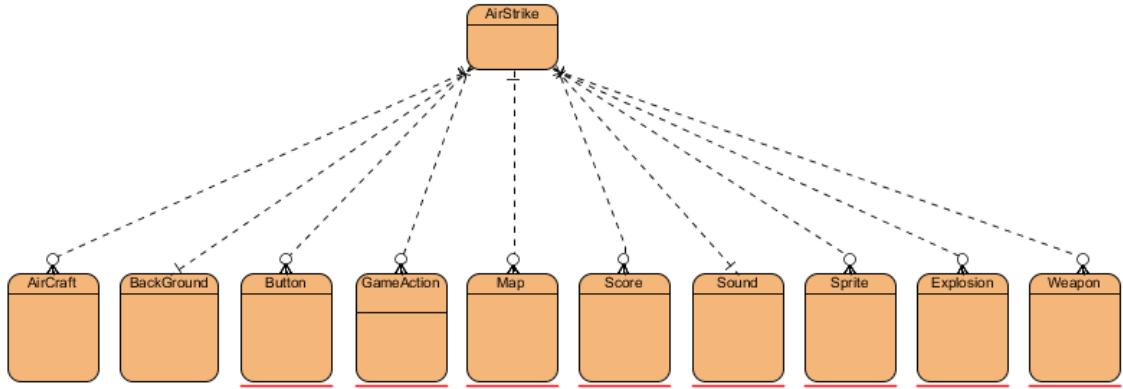


The sprite class is class which load image and store the images frame in to an *arrayList* frames. The class has two methods: *getFrame* and *addFrame*. It also provides the current sprite location. By using the *AnimFrame()* method, the class can get the image and time for each frame.



The main program is class AirStrike. This is a name of one city in the history. First, I need to define the constant variables such as map length, how many time players can hit the bullets or bombs, how much time the enemies hit the main character...

AirStrike
+void main(string [])
+int AirCraft(enemy)
+Explosion BigExplosion()
+Explosion SmallExplosion()
+void loadImage()
+void draw(Graphic2D)
+boolean TestCollision()
+void update_player_weapon()
+void update_enemy_weapon()
+void update_boss_weapon()
+void update_enemy_fly()
+void update_boss_fly()
+void set_enemy()
+void reset_game()
+void process_when_player_died()



Description of subclass on AirStrike class:

- *Void loadImages()*: This class load the image from directory, add frame and set images to new sprite. It loads main character, enemies, weapons and map images. Each image has its own sprite and location on the screen. After I have main character and enemies sprites, I set the properties for each one by recall **AirCraft** class. The player has sprite, move speed, fire time and location. This class also loads image for weapon system. Each kind of weapon has its own sprite, speed, damage and location.
- *Void update_player_weapon* : This class handles player weapon. The player weapons are bullet and bomb. Each time player use weapon, I will check collision between weapon and enemies and enemies' weapons also. I'm use setScore to set how much score player can earn for each type of enemies.
- *Void update_enemy_fly* : This class updates and check collision for fly enemies and player. The main character will die if it hit the enemies.
- *TestCollision()* : this class tests collision between two objects. Each object has sprite, move speed and location. Assume that each sprite is inside a rectangle.

Each rectangle has location and two properties are width and height. We need to check if two rectangles intersect or not. If yes, the return value will be true.

2.4.6: Screen graphic

There are two parts to display hardware. There are video graphic cards and monitors. The video card stores what is on screen by its own memories. It also controls what is displayed by modifying several functions. To make it simple, we can understand that the video card pushes everything in its memories to the monitor and the monitor displays the information that the video card tells it to.

The screen is divided into many tiny pixels, and the origin is in the upper-left corner. Each pixel has its own position and can be expressed as (x,y) where x is the number of horizontal pixels and y is the number of vertical pixels from the origin. Starting from the upper-left corner is a point has position (0,0), the screen display left to right, top to bottom. For example, if we have a screen where the resolution is 800x600, the pixel at the upper-right will be (799,0), the pixel on bottom-left is (0,599) and the bottom-right is (799,599). The resolution depends on video card and monitor. Some suggestions are 640x480, 800x600, 1024x768...

The second thing needs to take care about is pixel color and bit depth. There are three basic colors: yellow, blue and red. These are the primary colors. We can have other colors by mixing three primary colors. The monitor does the same what we do. It will combine red, blue and yellow to create any color. But, how much can the monitor display? The number of colors a monitor can display depends on the bit depth of the display mode. The common bit depths are 8,15,16,24 and 32. For example, 8-bit color has $2^8 = 256$ colors. That means only 256 colors can be displayed at a time. For 15-bit colors, the total colors can display is $2^{15} = 32,768$ colors. To make sure we can pick the

correct display mode to use, we need to make sure the game runs in at least two resolutions and using different bit depth to test.

3: TESTING AND RESULTS

3.1: Testing

Game testing is another important part of software development. This is a software testing process for quality and balance of video games. The goal of game testing is discovery and documentation. Feedback from players/testers was helpful to improving BlueSky.

BlueSky was tested with students on different operating systems. An evaluation form was used to ask students to play BlueSky and provide feedback. The form has ten questions about the game play and game ideas. For more details about the testing form, see appendix B. The students asked to play the game come from two groups. The first group is students who have experience playing video games. The second group is students who do not have much experience playing video games. The operating systems used to test the game were windows 7 and XP. The BlueSky tested in the different screen resolutions: 1600x900 and 1024x768.

3.2: Results

The results from the two groups of students were different. The second group played BlueSky longer than the first. They spent more than 20 minutes to play and tried to finish the game. Some students in Group Two tried to display their game play skills, attempting to dodge between enemies and bullets. Group One spent less time than Group Two. To help Group One can finish the game, I reduced speed of enemies and increased movement speed of main character and bullets also. There was some similar feedback from both groups. First, the consensus was that the game was a little short. Play testers wanted to have a longer map or more stages to play. Also, both groups estimated the target age range to be from 15-20 years old. Two students suggested the creation of a

campaign mode because they wanted to follow the game story. Some feedback recommended that the video game be able to run on more operating systems and with more screen resolutions.

The table below shows the result for the testing. There are two groups: Group One is non-experience and Group Two is experience players. Each group has ten testers. The following graphic present the testing results. Number 0-10 describe lowest bound to the highest bound.

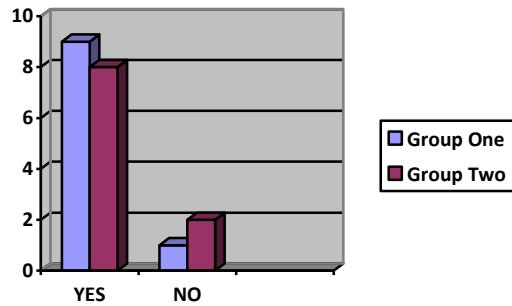


Figure 5: Response to the question “Do you like game”

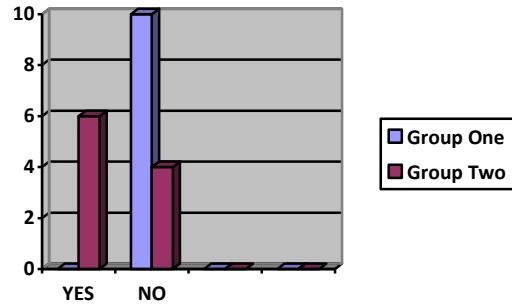


Figure 6: Response to the question “Did you finish the game”

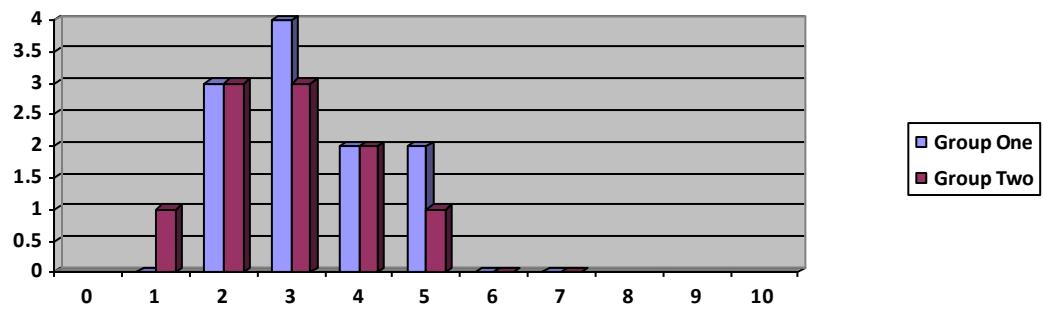


Figure 7: Response to the question “How many times you play game”

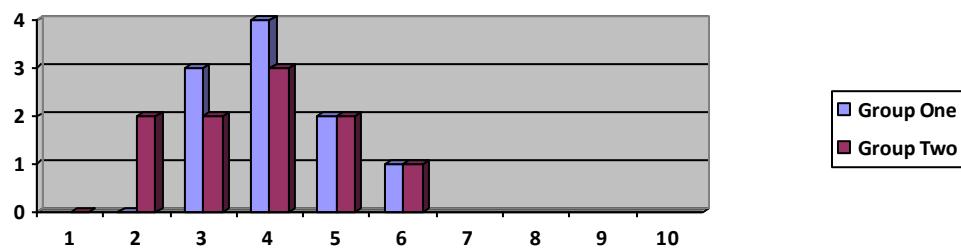


Figure 8: Response to the question “Was the game too short, too long or just right?”

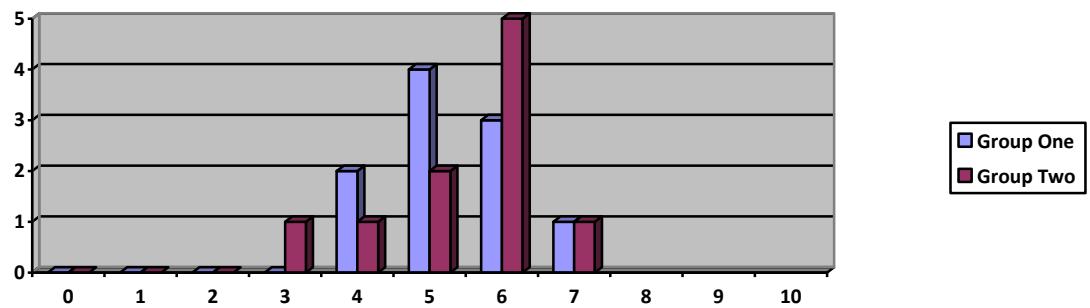


Figure 9: Response to the question “how much you like materials and/or game pieces”

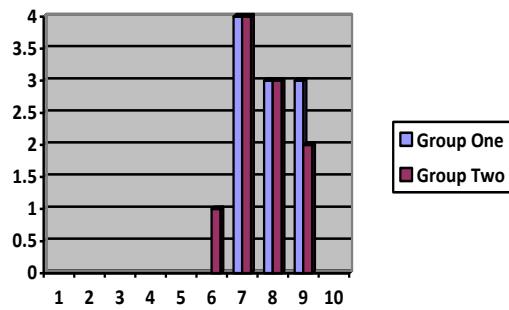


Figure 10: Response to the question "Do you like the game idea?"

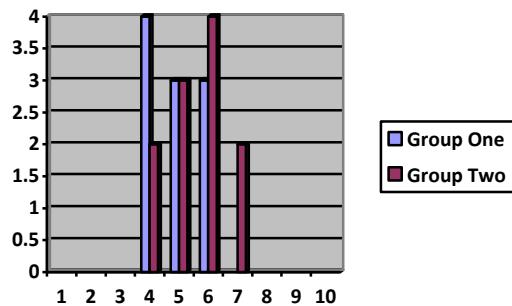


Figure 10: Response to the question "was text too small or big?"

These results prompted the creation of different difficulty levels. By increasing the player bullet speeds and ammunition supply, along with the number of player bombs, the less-experienced players were able to finish the game. In addition, the map length has been increased in response to Group Two so they can enjoy playing the game longer.

4: FURTURE WORK

4.1: Game Modes and Game Stages

There are two types of game mode: Campaign mode and Survival mode.

Campaign mode facilitates the player progressing through the story line of the game, while Survival mode challenges players to play/survive as long as possible. BlueSky's default mode is Survival mode. In addition to gameplay modes, BlueSky has three stages.

In order to make the game more interesting, more stages and enemy types are in development.

4.2: Game level

BlueSky has only one difficulty level now. Ideally, this will be expanded in the future to include the classic three difficulty levels: easy, normal, and hard. Also, the player should be able to choose which stage to play. The game design will require an update to present these new options to players in the main menu screen. The default difficulty level will be normal. The movement speed of enemies and bullets will increase as the player progresses stages. A player playing on easy mode should be much more difficult to defeat, and will have more ammunition with faster moving projectiles, while enemy bullet speeds are reduced.

4.3: Developing for mobile devices.

Mobile devices such as tablet and mobile phone are extremely popular these days; a normal part of most peoples' daily lives. The entertainment industry is growing as fast as technology. Developing this game for mobile platforms is a good way to expand the customer base, and make BlueSky accessible for more players.

REFERENCES

Battleship [Image]. (2014, August 26). Heavy weapon. Popcap game.

Brackeen, D., Bracker, B., & Vanhelsuwe, L. (2003). *Developing game in java*.

Davison, A. (2005). *Killer game programming in java* (B. McLaughlin, Ed.). Sebastopol, CA: O'Reilly Media.

Delta Fighter [Image]. (2014, August 26). Heavy weapon. Popcap game.

Driven Systems with Undetermined Input Spaces", *Software Engineering, IEEE Transactions on*, On page(s): 216 - 234 Volume: 40, Issue: 3, March 2014

Helicopter [Image]. (2014, August 26). Heavy weapon. Popcap game.

Jet-Fighter [Image]. (2014, August 26). Heavy weapon. Popcap game.

Kent, S. (2010). *The ultimate history of video games: From pong to pokémon--The story behind the craze that touched our lives and changed the world*.

Microsoft directX [Lecture notes]. (n.d.). Retrieved September 30, 2014, from Computer Hope website: <http://www.computerhope.com/directx.htm>

Nguyen, B.N.; Memon, A.M. "An Observe-Model-Exercise* Paradigm to Test Event *Other Built-in Functions Provided by GCC* [Lecture notes]. (n.d.). Retrieved October 5, 2014, from Gcc.gnu.org website:
<http://gcc.gnu.org/onlinedocs/gccint/Libgcc.html>

Pickup truck [Image]. (2014, August 26). Heavy weapon. Popcap game.

Sanchez Crepso Dalmau, D., & Engel, W. (2003). *Core techniques and algorithms in game programming*. New Riders.

Schach, S. R. (2007). *Object-oriented and classical software engineering* (8th ed.). Raghathan Srinivasan.

Selman, D. (2002). *Java 3D programming* (2nd ed.). Manning.

T-21 [Image]. (2014, August 26). Heavy weapon. Popcap game.

T-31 [Image]. (2014, August 26). Heavy weapon. Popcap game.

T-83 [Image]. (2014, August 26). Heavy weapon. Popcap game.

The GCC low-level runtime library [Lecture notes]. (n.d.). Retrieved October 5, 2014,

from Gcc.gnu.org website: <http://gcc.gnu.org/onlinedocs/gccint/Libgcc.html>

Where is the DirectX sdk? (n.d.). Retrieved October 9, 2014, from Microsoft MSDN

website: <http://msdn.microsoft.com/en-us/library/ee663275%28v=VS.85%29.aspx>

Cho, O.-H., & Lee, W.-H. (2011). *An Interactive Event-Design Tool for Rapid Game Development*, pp. 450-453. <http://dx.doi.org/10.1109/ISCE.2011.5973868>

Mao, C., Yi, Z., JianGang, O., & Guo, H. (2010). *Game Design and Development Based on Logical Animation Platform*, pp. 573-576.

<http://dx.doi.org/10.1109/ICCIS.2010.146>

Fahy, R., & Krewer, L. (2012). *Using Open Source Libraries in Cross Platform Games Development*, pp. 1-5. <http://dx.doi.org/10.1109/IGIC.2012.6329835>

Popcap game group. (n.d.). New files. Retrieved October 7, 2014, from Heavy Weapon Wiki website: <http://heavyweapon.wikia.com/wiki/Special:NewFiles>

Appendix A: GAME DESIGN

1: Game Mechanics

The game mechanics are constructs designed for interaction of the video game. Every video game uses mechanics. The difference in game mechanic between every game is game design. In the BlueSky, each character or enemy has its own life points and weapon. The other name of life point can be hit or health points (HP). Life point is an integer number which describes how much damage a character or enemy can withstand.

1.1: Character:

As a pilot, player will control a helicopter to fight with enemies.



Figure 11: Main character

Life point: 5

Weapon: Bullet and Bomb

The character has three life points. Each time player is hit by any enemies, the life points will reduce by one. Player needs to go through three stages and defeat a boss to end the video game. The main character has its own weapons. Because we have two kinds of enemies: ground and fly, so the characters have two kinds of weapons to kill each kind of enemies.

1.2: Enemies

There are two kinds of enemies: Ground and Fly enemy. In addition, the game has a special enemy is boss.

1.2.1: Ground enemies

Ground enemy is standing in the ground. Its weapon is bullet. This enemy hits player by the bullet from the ground.

Life point: 1

Weapon: Bullet

Damage to player: 1



Figure 12: Ground enemy

1.2.2: Fly enemies

Fly enemy is a flying object. It can stand alone or more depend on the setting. It can hit player by bullet.

Hit point: 1

Weapon: Bullet.

Damage to player: 1

Each fly enemy can appear random to player on the map.

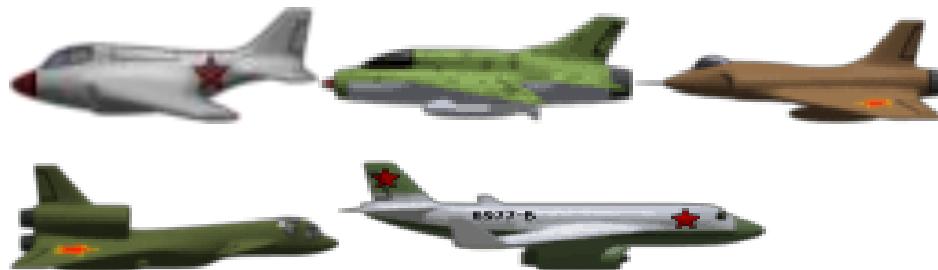


Figure 13: Fly enemies

1.2.3: Boss

Boss is special enemy. The boss has more life points and weapons. The sprite is display boss like a mothership.

Hit points: 5

Weapon: Bullet

Damage to player: 1



Figure 14: Boss

1.2.4: Weapons



Figure 15: Bullet

2. Explosion:

There are two explosions screen in game. There are small_explotion and big_explosion .

The small_explition does appear when player hit enemies' rocket and the big_explosion does appear when player hit the enemies.

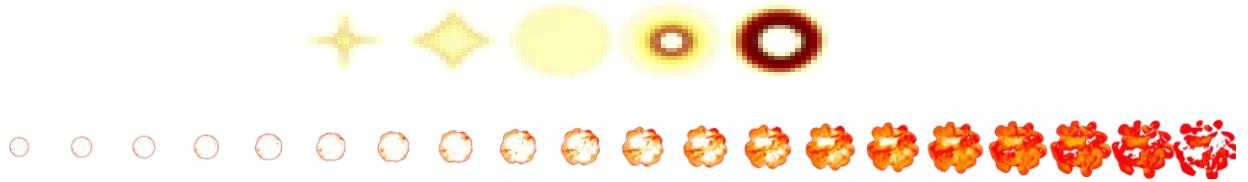


Figure 16: Explosion

3: Scoring system

The scoring system will count score of player. When player hit an enemy, they will earn a constant score number. The Scoring system will store this number and when it counts until 1000, the player life points will increase by one. In addition, the enemies not only can hit the enemies but also hit the enemies' bullets. Each time player hit the enemies' weapon (Bullet); they earn 10 points score number. For example, the enemy hit a bullet to player, if player hit a bullet to this rocket, he will earn 10 points. In addition, if player hit and kill one enemy, he will earn 40 point. We do store the player score and display it to the action screen.

4: Music

Each type of game action has different sound.

The list below is a soundtrack of this game.

Bonus sound: Play when player get 1000 score and life point increase by one

Victory sound: Victory

Enemy missile: when an enemy hit a bullet

Player missile: When player hit a bullet

Explode: enemy or player die. In addition, when player hit enemy's bullet, the sound is different with hit enemy. The sound when player hit enemy bullet is small than the sound when player hit enemy.

Bomb fall: when player bomb ground enemies.

Themes song: the background music.

Appendix B: USER MANUAL

1: Game controller

Gamer need to control the character by use arrow down, up, left, right in the keyboard. During playing time, player can hit “Esc” to pause the game and the program will display a small window for three options: Esc to continue game, Q to quit to main menu. To attack the enemies, the player hit “Ctrl” or “Shift” to shoot by bullet or bomb for the flight enemies and ground enemies.

There is the list of game controller using keyboard:

- ESC: Pause game when player is playing game, and Continue when player is paused game
- Q: Quit, Return main menu
- Ctrl: Shoot
- Shift: Bomb
- Key up: move up
- Key down: Move down
- Key left: move left
- Key Right: move right
- C: Continue

2: Game play

2.1 Start scene

At the beginning, the game will load the game’s background and display game menu. The game menu is three functions: Start game, Game guide and Exit game. The game will begin when player click on the “start game” icon. Therefore, me screen will move to the action scene.

2.2: Action scene

In the left top of the action screen, there will be the player life points, and next to it will be player score.

The action scene will be a top and bottom background. When I put two-image overlap together, I can take random background that match with random bottom image to make video game screen.

2.3: Help scene

Help scene is the video game introduction. This is place where I putted the details information that need for gamer to understand how to play the game. The help scene is a combination of keyboard function and video game description. It will show the function for each keyboard that we need to use to play the game.

Appendix C: TESTING FORM

1: How much score do you have?

<1000 1000-2000 2000-2500 2500-5000 More than 5000

2: What age range do you think this game is suitable for?

1-10 10-15 15-20 20-25 25-30 Over 30

3: Did you finish the game?

Yes No

4: Do you like the game?

Yes No

5: How many times did you play the game?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

6: Was the game too short, too long or just right?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

Too short Just right Too long

7: How much did you like the materials and/or game pieces?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

Did not like Average Loved

8: Game Idea

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

Weak OK Terrific

9: Was the text on the instructions too small or just right?

0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

Too small Just right Too big

10: Any suggestion?